Восход PostgreSQL на Эльбрус

PgConf 2017

ФГБУ НИИ «Восход: Чижевский Игорь Евгеньевич
Мерзляков Станислав Николаевич
Погибенко Дмитрий Алексеевич
Богданов Иван Владимивирович

АО «МЦСТ»: Королёв Сергей Дмитриевич

Data Egret (PostgreSQL-Consulting): Космодемьянский Илья Андреевич

Внедрение PostgreSQL на Эльбрусах в систему паспортно-визовых документов нового поколения

О чём расскажем:

- Кто мы такие, что мы делаем и зачем?
- Перевод кровавого Enterprise-приложения на PostgreSQL
- Миграция данных на живой системе
- Выявление проблем производительности PostgreSQL на Эльбрусе
- Адаптация PostgreSQL под платформу
- Как эксплуатируем (мониторинг и отказоустойчивость)

Чижевский Игорь

Архитектор
ФГБУ «НИИ Восход»

ГС МИР / ГС ПВДНП: что это?

- ГС ПВДНП: загранпаспорт нового поколения и не только
- Территориально-распределённая система с ЦОДами, главный — на Восходе
- Что делаем мы

ГС МИР / ГС ПВДНП: чем примечательна?

- Linux и свободное ПО от начала времён
- Защищённость по требованиям ФСБ России и ФСТЭК, тотальная электронная подпись везде
- PostgreSQL: уже несколько лет успешно работает в Системе
- Первые на Эльбрусах!

ГС МИР / ГС ПВДНП: в импортозамещение с головой!

С чего всё начиналось, что было и что стало

Что было:

- Мейнфреймы
- IBM DB2 for z/OS
- IBM WebSphere MQ
- IBM Tivoli

Что стало:

- Эльбрусы и другое
- PostgreSQL, Ceph
- Apache ActiveMQ, Redis
- Zabbix, OTRS и другое

Приклад: Java (Tomcat, Spring, Hibernate, Camel и так далее)

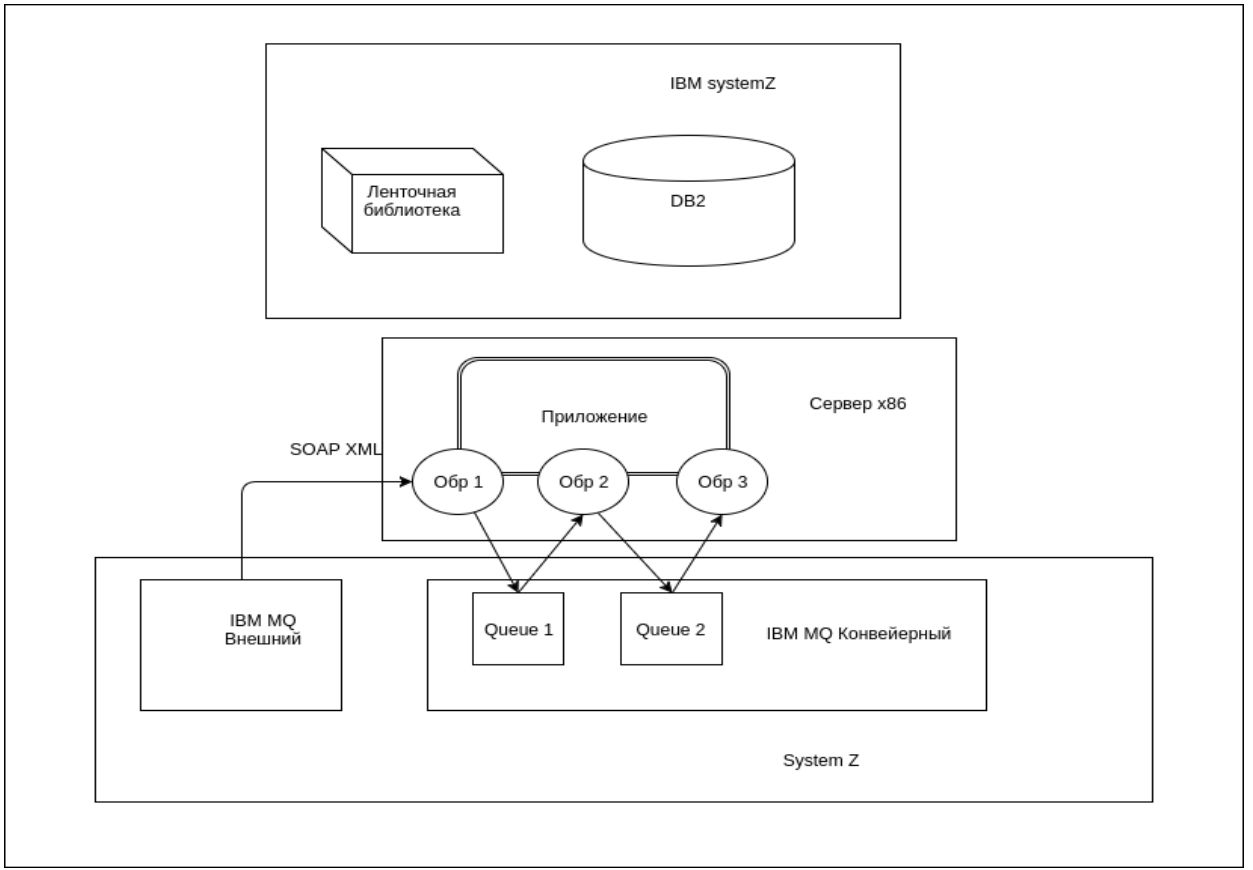
Особенности перевода приложения на PostgreSQL

Мерзляков Стас

Руководитель группы разработки
ФГБУ «НИИ Восход»

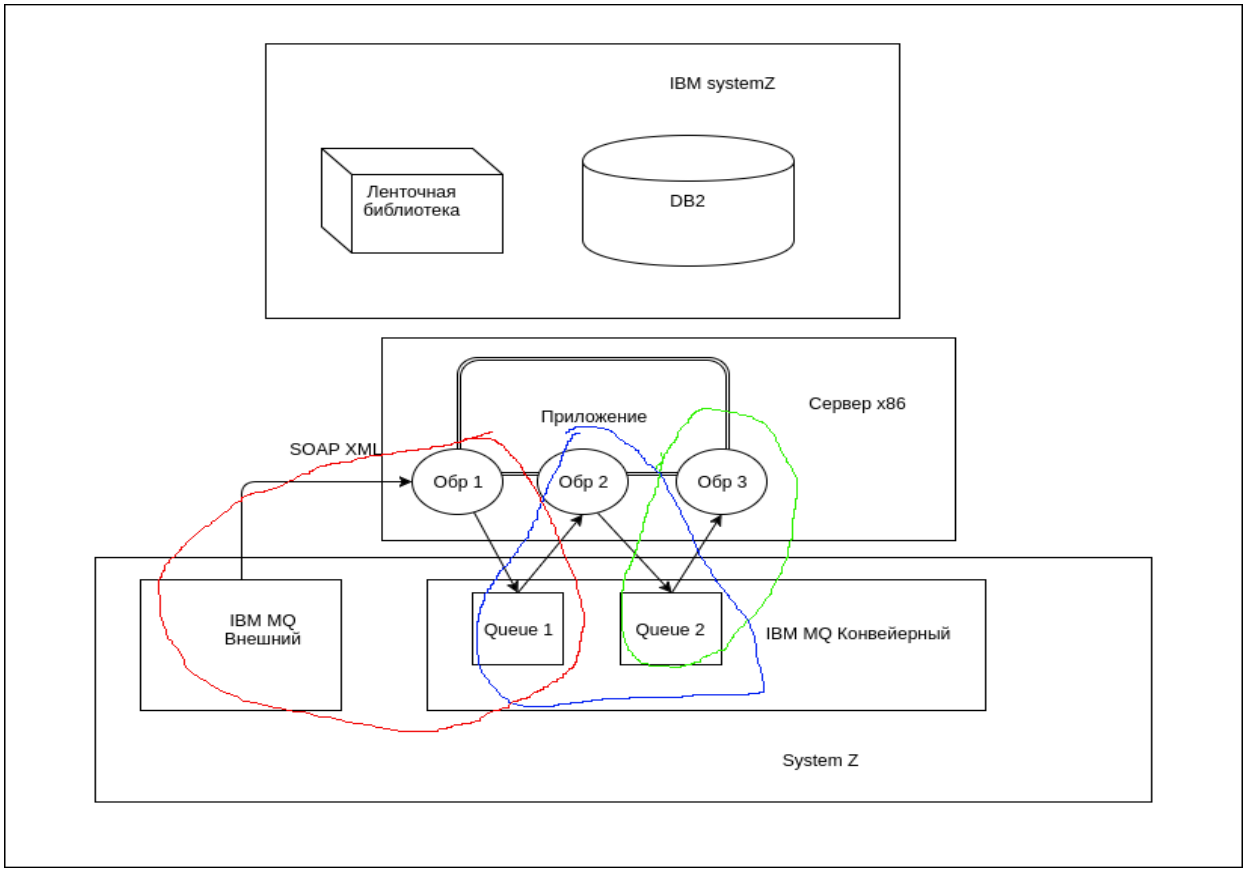
Особенности перевода приложения на PostgreSQL

- Что было: общая архитектура



Особенности перевода приложения на PostgreSQL

- Что было: двухфазные транзакции



Особенности перевода приложения на PostgreSQL

- Перенос блобов в отдельное хранилище

- Вынести блобы из базы данных
- Вынести блобы из сообщений
- Кодовая база должна остаться единой
- ЭП XML сообщений не должна испортиться

Особенности перевода приложения на PostgreSQL

- Перенос блобов в отдельное хранилище

- Ссылка на базе хэш
- В теле сообщения заменяем блоб на строку base64(Ref)
- В БД пишем просто Ref
- JPA @EntityListener для отслеживания изменений

Особенности перевода приложения на PostgreSQL

- Idle in transaction

- DB2/zOS — блокировочная база, держит много транзакций
- PostgreSQL — версионная база, проблемы

Особенности перевода приложения на PostgreSQL

- Idle in transaction

- Как проверить что что-то не так (немного магии от Data Egret) :
- select query from pg_stat_activity where state='idle in transaction' and state_change < now () - '0.1 seconds'::interval
- show pools в pgbouncer (cl_waiting)

Особенности перевода приложения на PostgreSQL

- Idle in transaction

- Там где получилось — вынесли работу с БД в отдельные транзакционные методы
- Там где не получилось — разделили логику на несколько обработчиков (через конвейер)

Особенности перевода приложения на PostgreSQL

- Двухфазные транзакции

- Внешнее взаимодействие: протокол взаимодействия объектов между собой предполагает возможность переотправки сообщений
- Конвейер: Требуется что бы одно сообщение обрабатывал в данный момент только один обработчик
- Конвейер: Обработка может быть повторена сколько угодно раз. При повторных вызовах изменений в БД больше не производится (за этим следит сам обработчик)

Особенности перевода приложения на PostgreSQL

- Двухфазные транзакции

- Заменили СГДС на Redis(ZSET)
- Добавили журнал прохождения по конвейеру
- Написали свой Apache Camel компонент который берет всю работу с журналом и Redis на себя

Особенности перевода приложения на PostgreSQL

- Вопросы архитектуры: как делить монолит на микросервисы

Осторожно грабли!!!

Идея: а давайте просто разделим приложение на функциональные части и запустим каждую часть отдельно!

Особенности перевода приложения на PostgreSQL

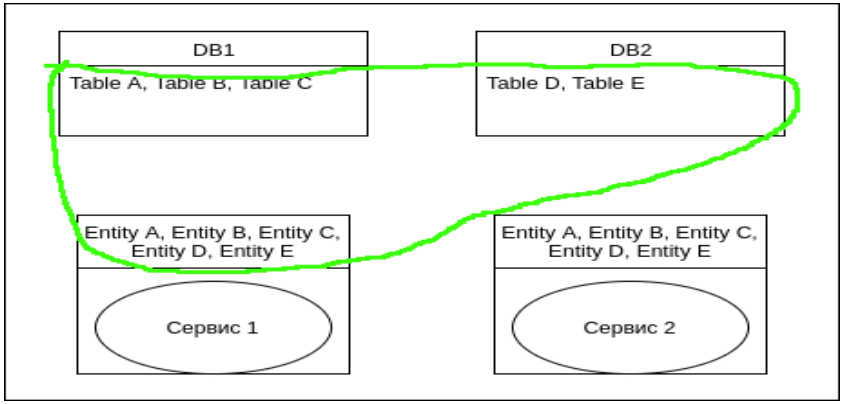
- Вопросы архитектуры: как делить монолит на микросервисы

- Легко выделяется функционал не использующий БД (либо изменения в котором нам не критичны)
- Сильная связность между бинами и сущностями
- Разделение на части приводит к проблеме распределенных транзакций

Особенности перевода приложения на PostgreSQL

- Вопросы архитектуры: как делить монолит на микросервисы

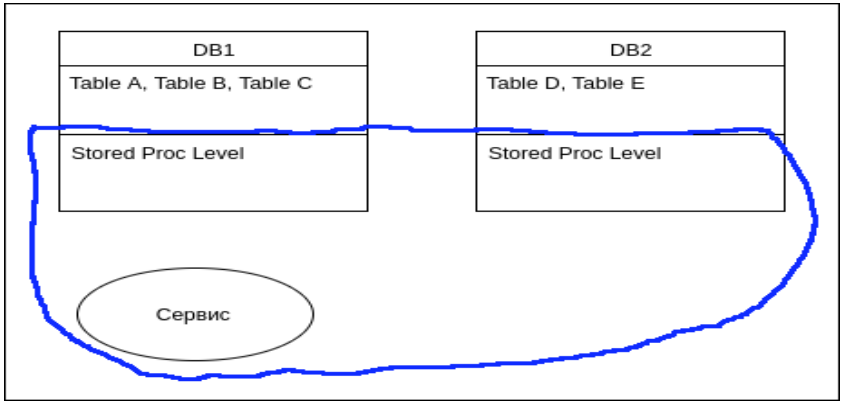
Синхронизация на уровне Hibernate. В persistence.xml сервисов должна быть информация о @Entity всех БД.



Особенности перевода приложения на PostgreSQL

- Вопросы архитектуры: как делить монолит на микросервисы

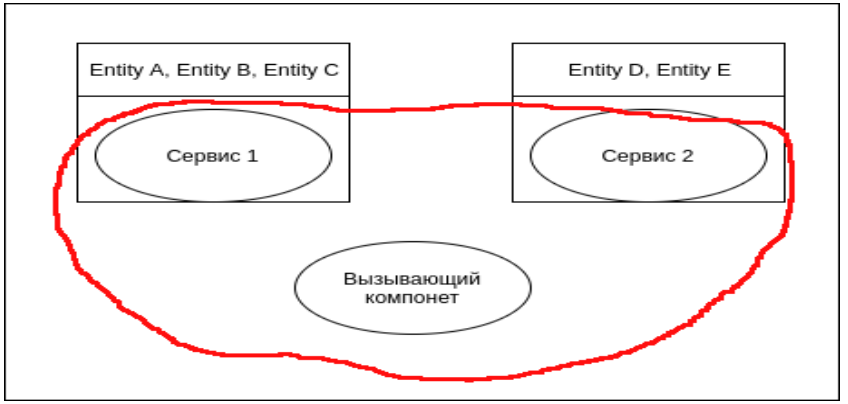
- Синхронизация через хранимые процедуры. Часть логики уезжает в БД



Особенности перевода приложения на PostgreSQL

- Вопросы архитектуры: как делить монолит на микросервисы

- Интеграция через XA-web-сервисы. Web Services Transaction (WS-TX)



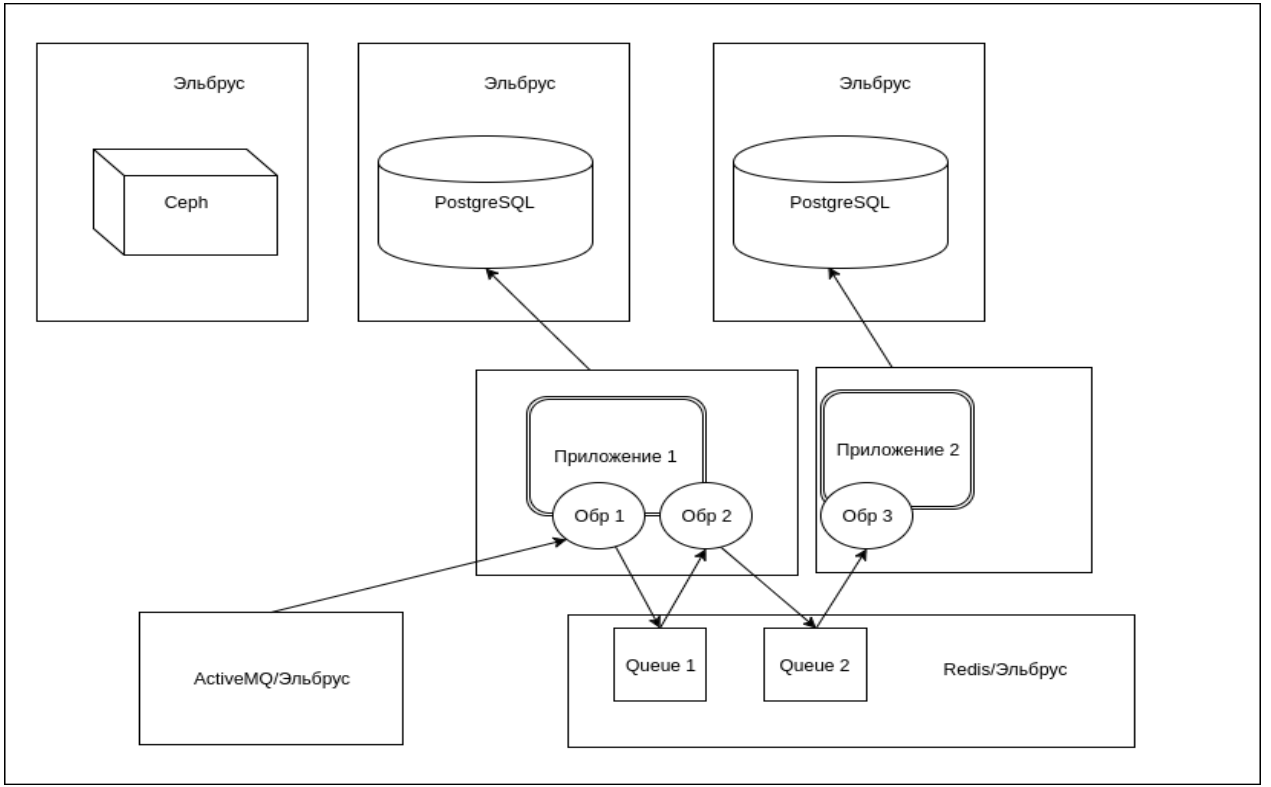
Особенности перевода приложения на PostgreSQL

- Вопросы архитектуры: как делить монолит на микросервисы

- Одно сообщение обрабатывает в данный момент только один обработчик
- Обработка может быть повторена сколько угодно раз. При повторных вызовах изменений в БД больше не производится (за этим следит сам обработчик)
- Обработчик может вносить изменения только в свою функциональную часть
- Информация в других частях доступна только в read-only режиме (либо какие-то изменения нам не критичны)

Особенности перевода приложения на PostgreSQL

- Что получаем



На живую нитку
(как сменить БД на работающей Системе)

Погибенко Дмитрий

Ведущий разработчик
ФГБУ «НИИ Восход»

Требования к миграции

- **Трансформация данных при переносе**
- **Перенос данных без остановки БД**
- **Зависимости между таблицами**
- **Перенос данных в сжатые сроки**
- **Верификация данных после переноса**

Требования к миграции

- **Трансформация данных при переносе**
- **Перенос данных без остановки БД**
- **Зависимости между таблицами**
- **Перенос данных в сжатые сроки**
- **Верификация данных после переноса**

Требования к миграции

- **Трансформация данных при переносе**
- **Перенос данных без остановки БД**
- **Зависимости между таблицами**
- **Перенос данных в сжатые сроки**
- **Верификация данных после переноса**

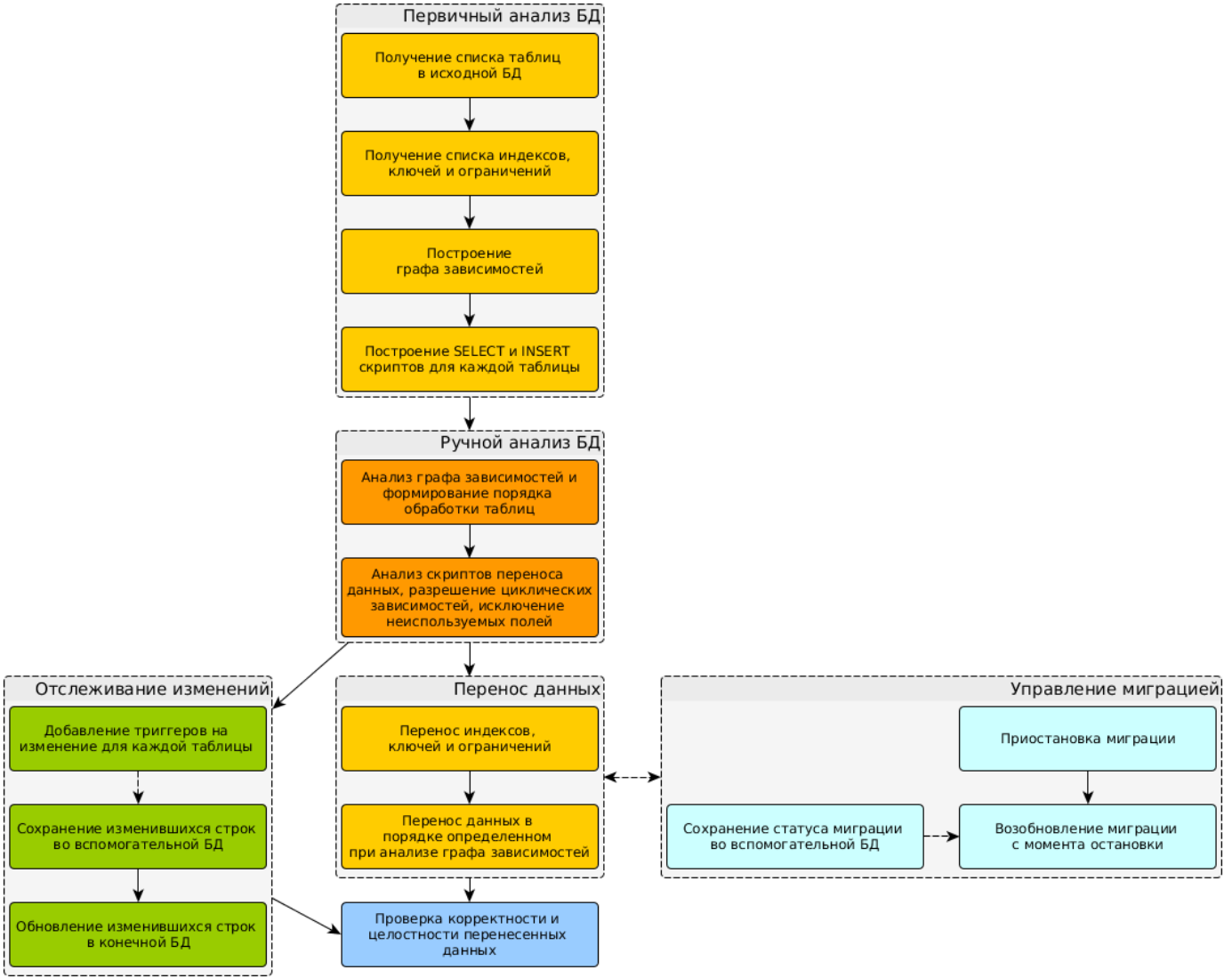
Требования к миграции

- **Трансформация данных при переносе**
- **Перенос данных без остановки БД**
- **Зависимости между таблицами**
- **Перенос данных в сжатые сроки**
- **Верификация данных после переноса**

Требования к миграции

- **Трансформация данных при переносе**
- **Перенос данных без остановки БД**
- **Зависимости между таблицами**
- **Перенос данных в сжатые сроки**
- **Верификация данных после переноса**

Схема процесса переноса данных



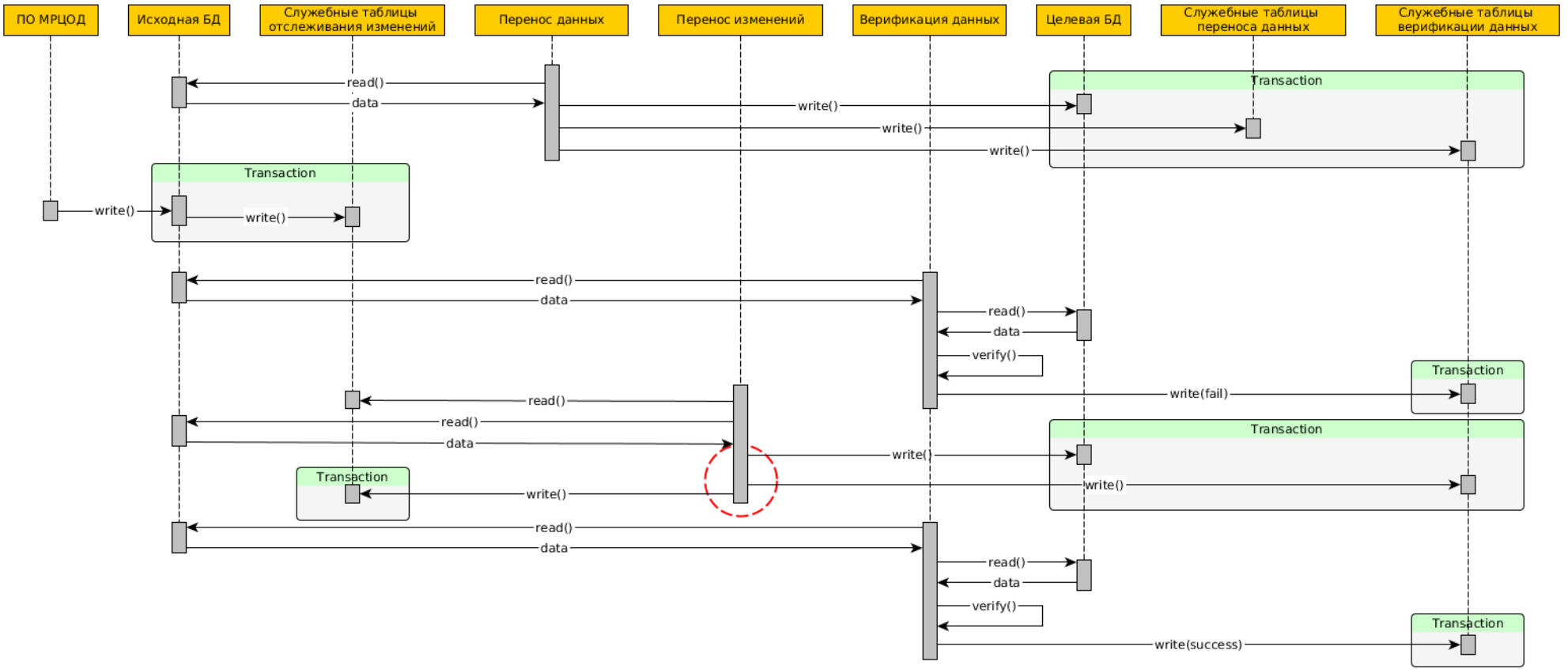
- **Думатель думает**
- **Анализатор анализирует**
- **Внутре неонка**

- **Триггеры отслеживают изменения**
- **SchemaSpy анализирует зависимости**
- **Spring Batch переносит данные**

- **Триггеры отслеживают изменения**
- **SchemaSpy анализирует зависимости**
- **Spring Batch переносит данные**

- **Триггеры отслеживают изменения**
- **SchemaSpy анализирует зависимости**
- **Spring Batch переносит данные**

Как всё работает, включая транзакции



Выводы

- **«Серебряной пули» нет**
  Все зависит от большого числа факторов.

- **Используйте существующие инструменты, а не изобретайте велосипед**
  Хороший разработчик — ленивый разработчик

- **Ищите баланс**
  Чем выше надежность, тем ниже производительность.
  А вот обратное не всегда верно

Эльбрус с точки зрения администратора PostgreSQL

Опасения

Эксплуатация СУБД на новой платформе всегда вызывает беспокойство у DBA

Почему?

- СУБД интенсивно использует низкоуровневый функционал ОС для обеспечения производительности
- Для СУБД надежность аппаратного обеспечения имеет очень высокий приоритет
- Насколько качественно работает поддержка платформы производителем?
- «Большим данным – быстрые диски!» - что у Эльбруса с производительными дисками, контроллерами, памятью?

Первые впечатления: Эльбрус ОС

- В целом – Linux! Уже хорошо, эксплуатация PostgreSQL под Linux - хорошо проработанный вопрос
- GCC-совместимый компилятор. Есть perf!
- Но ОС имеет свои особенности в деталях: например, пользователь который может запустить демона, не может иметь crontab!

42 Попробуем запустить PostgreSQL

Главный вопрос, который интересует на этом этапе:

Какую нагрузку выдержим по сравнению с привычными платформами?

Первый блин… Huge pages

- Для экономии на TLB большие объемы памяти (shared_buffers ~>8Gb) выделяют большими чанками
- Оперировать huge pages должна уметь ОС и user space приложение
- Сборка PostgreSQL под Эльбрус ОС не может… Идем тестировать поддержку от производителя!
- 2 недели – новый релиз сборки, huge pages заработали, переходим к более серьезным тестам

44 Сравниваем 16 ядер

- Основной вопрос – как ведет себя база на Эльбрусе в условиях, приближенных к OLTP
- Подбираем аналогичный AMD сервер и тестируем против него Эльбрус

Кол-во соеденений	TPS AMD	TPS Эльбрус	Разница
1	12.200	3.600	3.4x
16	187.000	53.000	3.6x
32	180.000	47.500	3.8x
64	195.000	41.500	4.7x
128	195.000	35.300	5.6x
256	183.000	29.500	6.2x

Размер базы 10Gb, простой тест readonly pgbench

Что-то тут не так!

AMD

```
3,34%  postgres                [.] FunctionCall2Coll
 2,80%  postgres                [.] AllocSetAlloc
 2,63%  postgres                [.] hash_search_with_hash_value
 2,24%  libc-2.23.so            [.] _int_malloc
 2,18%  libc-2.23.so            [.] vfprintf
 1,88%  postgres                [.] PostgresMain
 1,80%  postgres                [.] _bt_compare
 1,66%  postgres                [.] btint4cmp
 1,57%  postgres                [.]SearchCatCache
 1,48%  libc-2.23.so            [.] __memcpy_sse2_unaligned
 1,44%  postgres                [.] hash_any
 1,19%  libc-2.23.so            [.] _int_free
 1,19%  [vdso]                  [.]__vdso_gettimeofday
 1,09%  postgres                [.] LockReleaseAll
 1,05%  libc-2.23.so            [.] __memcmp_sse2
 1,02%  postgres                [.] AllocSetFree
 ...
 0.95%  postgres                [.] LWLockAttemptLock
 ...
 0.22%    1,03%  postgres        [.] PinBuffer
```

Эльбрус

```
 5,93%  postgres          [.] _bt_compare
3,82%  pgbench           [.] dopr
2,26%  libc-2.23.so      [.] _int_malloc
2,03%  postgres          [.] hash_search_with_hash_value
1,68%  libc-2.23.so      [.] _int_free
1,61%  postgres          [.] LWLockAttemptLock
1,59%  libc-2.23.so      [.] malloc
1,56%  postgres          [.] AllocSetAlloc
1,46%  postgres          [.] SearchCatCache
1,45%  libc-2.23.so      [.] __gettimeofday
1,41%  libc-2.23.so      [.] malloc_consolidate
1,35%  postgres          [.] dopr
1,28%  postgres          [.] LockReleaseAll
1,18%  pgbench           [.] doCustom
1,07%  postgres          [.] hash_any
1,04%  pgbench           [.] threadRun
1,03%  postgres          [.] PinBuffer
```

Разница

```
AMD    Эльбрус
 1.80%  5,93%  postgres [.] _bt_compare   #нет вызовов btint4cmp? 0.95%  1,61%
 postgres [.] LWLockAttemptLock
 ???%   3,82%  postgres [.] dopr
 0.22%  1,03%  postgres [.] PinBuffer
```

- _atomic* работают пропорционально частоте CPU – у Эльбруса она меньше
- Размер кэшей на Эльбрусе меньше
- Но разница все равно очень большая – требуется оптимизация LWLock на уровне сборки PostgreSQL
- 1 месяц до нового релиза… LWLocks в сборке реализованы по образцу ARM

Выводы

- Эльбрус вполне достойно работает как платформа для PostgreSQL
- Дистрибутив PostgreSQL под Эльбрус ОС это Upstream:
  - модификации минимальны и касаются только платформенно-специфических ускорений.
  - Такую сборку проще синхронизировать
- Приятно радует оперативность реакции разработчиков на результаты тестирования

МЦСТ ЭЛЬБРУС

PostgreSQL на вычислительных комплексах семейства «Эльбрус»

Королев Сергей Дмитриевич

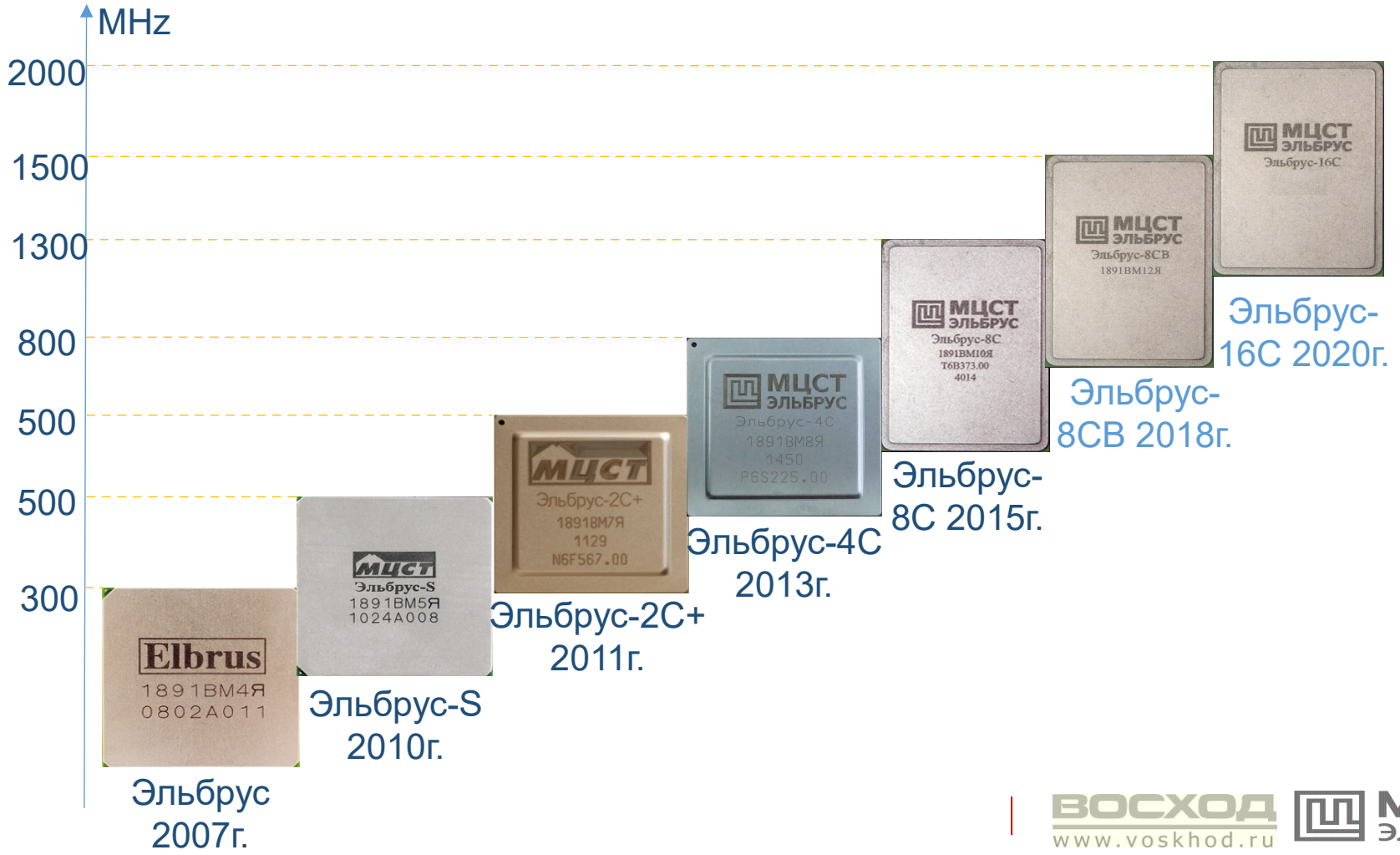
АО «МЦСТ»

16 марта 2017

О компании МЦСТ

- Разработка ключевых компонентов компьютера
  - Процессоры
  - Контроллеры
  - BIOS
  - Ядро ОС
  - Дистрибутив ОПО «Эльбрус»
- Собственные средства разработки
- Доверенная платформа

Эволюция процессоров Эльбрус



PostgreSQL в ОПО Эльбрус

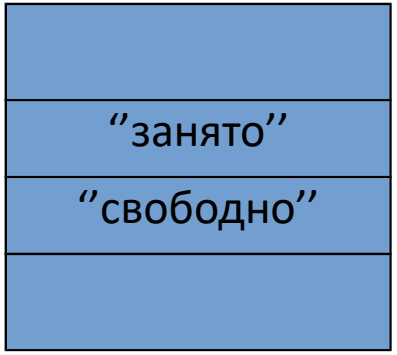
- PostgreSQL-9.5.2 (9.2.3, 9.6.2)
- Postgres-xl-9.5 (rc)
- Gpdb-4.3

Адаптация кода

- Архитектурно-зависимые части
- Код, требующий оптимизации для платформы Эльбрус
- Настройка кросс-компиляции

Архитектурно зависимые части

```
#if defined (__e2k__)
#define HAS_TEST_AND_SET
#define TAS(lock) tas(lock)
typedef int slock_t;
static __inline__ int
tas(volatile slock_t *lock)
{
    return __sync_lock_test_and_set(lock, 1);
}
#define S_UNLOCK(lock) __sync_lock_release(lock)
#endif
```



Sync_lock_test_and_set, sync_lock_release

```
register <val_type> prev_val, value, *addr;
asm volatile
 ( "1: <ld>,0   %[addr], %[prev], mas = 0x7"
  "   {"
  "    <st>,2  %[addr], %[value], mas=0x2"
  "    rbranch 1b"
  "   }"
  : [prev] "=&r" (prev_val), [addr] "+m" (*addr)
  : [value] "r" (value)
  : "memory");
return prev;
```

```
register <type> *addr, value;
  asm volatile
   ("wait ld_c = 1, st_c = 1"
    "<st> %[addr], %[value]"
    : [addr] "=m" (*addr)
    : [value] "r" (value)
    : "memory");
```

dopr vs vfprintf

```
 3,05% postgres   [.] LWLockAttemptLock
 2,53% postgres   [.] _bt_compare
 1,85% [kernel]   [k] __schedule
 1,83% libc-2.23.so   [.] _int_malloc
 1,60% pgbench    [.] dopr
 1,57% libc-2.23.so   [.] malloc
 1,55% postgres   [.] hash_search_with_hash_value
 1,28% postgres   [.] PinBuffer
 1,26% pgbench    [.] doCustom
 1,24% postgres   [.] dopr
```

Tps(no dopr) = 34207
Tps(dopr) = 35502
Улучшение на 4%

```
2,38% postgres   [.] LWLockAttemptLock
2,32% postgres   [.] _bt_compare
1,66% libc-2.23.so   [.] _int_malloc
1,60% libc-2.23.so   [.] malloc
1,34% postgres   [.] LWLockRelease
1,29% [kernel]   [k] __schedule
1,25% libc-2.23.so   [.] _int_free
1,14% postgres   [.] hash_search_with_hash_value
1,01% libc-2.23.so   [.] vfprintf
1,01% pgbench    [.] doCustom
```

Подсказки компилятору

- Атрибуты
- Builtin’ы
- Подсказки о маловероятных альтернативах

Восстановление БД из CSV

Первичный запуск

31,76% 9247 postgres postgres [.] CopyReadLineText
9,42% 2751 postgres postgres [.] CopyReadAttributesCSV
4,89% 1450 postgres libc-2.23.so [.] __GI_____strtoll_l_internal

-O4 + __builtin_expect

25,59% 7570 postgres postgres [.] CopyReadLineText
9,46% 2926 postgres postgres [.] CopyReadAttributesCSV
5,24% 1581 postgres libc-2.23.so [.] __GI_____strtoll_l_internal
4,90% 1447 postgres postgres [.] pg_verify_mbstr

0,"NS1DOSI",27244,618489464,485494816,"ZBRU304PFO","R6Z2H159QQ3IUCVT3Y726C33WW96RUL97AHEIDTE9NM9P7SYBP6
NRZWD2VAJG1IT16XP9GTO9LN4I69Z15CJVY230ICM915HX8WCUW6U7JOWV3LNZ4CK8KTETV6821V5FYN0KJ0X9UJUOBND5P
3JGMOFNKDWSFRD3L3TU9GTU6UO77RINK7TC1YPSHBA29TBKNUK2H4MDO0QLXEENBDP3HLL521DHKFRDFI6MCY565LXS5I
M71H09SRK1Y","N8886CEUOQWC287DB69JIMBPSQO9GVX99B8LTC5787Q05N47ZJG7BX3TTH8ZIBZHCXTWFO9DMP4XJEU83
H64KFN421TA2YHK1GM6VMQNHKAQ4BPXI1RS6K2YB2E3QLDYSQATIQMP026A31XRSUQ44I8LACU73EV1AVKICWDI59ZYGMG
E7C8HKMT1OTEYXZPX0GLIIP7D4W2BOOFLQU9G68N1RRPFW4I3AUBJ9O2JAUK5OQG4AVRMT4DR28","COJKHZKZOZ",630917
704,"2015-12-12","11:05:04","2016-10-17 12:39:10.412638",1795,51669998,812159288,"2015-12-12","12:11:04","2016-10-17
12:39:10.412638",23484

Time = 2m5s
Time(-O4 + __builtin_expect) = 1m53s
Ускорение на 9,6 %

__builtin_prefecth для tuplesort_heap_siftup

```
static void tuplesort_heap_siftup(Tuplesortstate *state, bool checkIndex){
SortTuple  *memtuples = state->memtuples;  SortTuple  *tuple; int i,n;
if (--state->memtupcount <= 0)
  return;
CHECK_FOR_INTERRUPTS();
n = state->memtupcount;
tuple = &memtuples[n];
i = 0;
for (;;){
  int j = 2 * i + 1;
  if (j >= n)
    break;
  if (j + 1 < n && HEAPCOMPARE(&memtuples[j], &memtuples[j + 1]) > 0)
    j++;
  if (HEAPCOMPARE(tuple, &memtuples[j]) <= 0)
    break;
  memtuples[i] = memtuples[j];
  i = j;
}
memtuples[i] = *tuple;}
```

```
__builtin_prefetch(&(memtuples[2*j+1]));
__builtin_prefetch(&(memtuples[2*j+2]));
```

Оптимизация синхронизации

Ускорение LWLockAttemptLock(Оптимистический подход к захвату LWLock)
https://habrahabr.ru/company/postgrespro/blog/270827/
Авторы: Александр Коротков, Дмитрий Васильев, Юрий Журавлев и возможно другие…

Было:

```
static bool LWLockAttemptLock(LWLock *lock, LWLockMode mode){
…
        old_state = pg_atomic_read_u32(&lock->state);
        while (true)        {
…логика
                if (pg_atomic_compare_exchange_u32(&lock->state,&old_state, desired_state))
 …
```

Стало:

```
static bool LWLockAttemptLock(LWLock *lock, LWLockMode mode)        {
…
        if (__builtin_expect((mode == LW_SHARED),1)){
                old_state = pg_atomic_fetch_add_u32(&lock->state, LW_VAL_SHARED);
                if (old_state & LW_VAL_EXCLUSIVE){
                        old_state = pg_atomic_sub_fetch_u32(&lock->state, LW_VAL_SHARED);
                        return true;
                } else
                {
                        return false;
                }
        }
        else if (mode == LW_EXCLUSIVE)
```

prefetchw руками

```
typedef struct LWLock
{
        slock_t         mutex;                  /* Protects LWLock and queue of PGPROCs */
        uint16          tranche;                /* tranche ID */
        pg_atomic_uint32 state;                 /* state of exclusive/nonexclusive lockers */
        dlist_head      waiters;                /* list of waiting PGPROCs */
        int garbage;
} LWLock;
```

Поправленные функции:
BufferAlloc (newPartitionLock->garbage=1;)
LockBuffer (buf->content_lock->garbage=1;)
GetSnapshotData (ProcArrayLock->garbage=1;)
LWLockAcquire (lock->garbage=1;)

Результат ускорения:
Pgbench(без правок)- 63790 tps
Pgbench(с правками) - 67644 tps
ускорение на 5,7%

Результаты профилирования

Первично:

```
4,31%  postgres        [.] GetSnapshotData
3,50%  postgres        [.] LWLockAttemptLock
3,08%  postgres        [.] _bt_compare
```

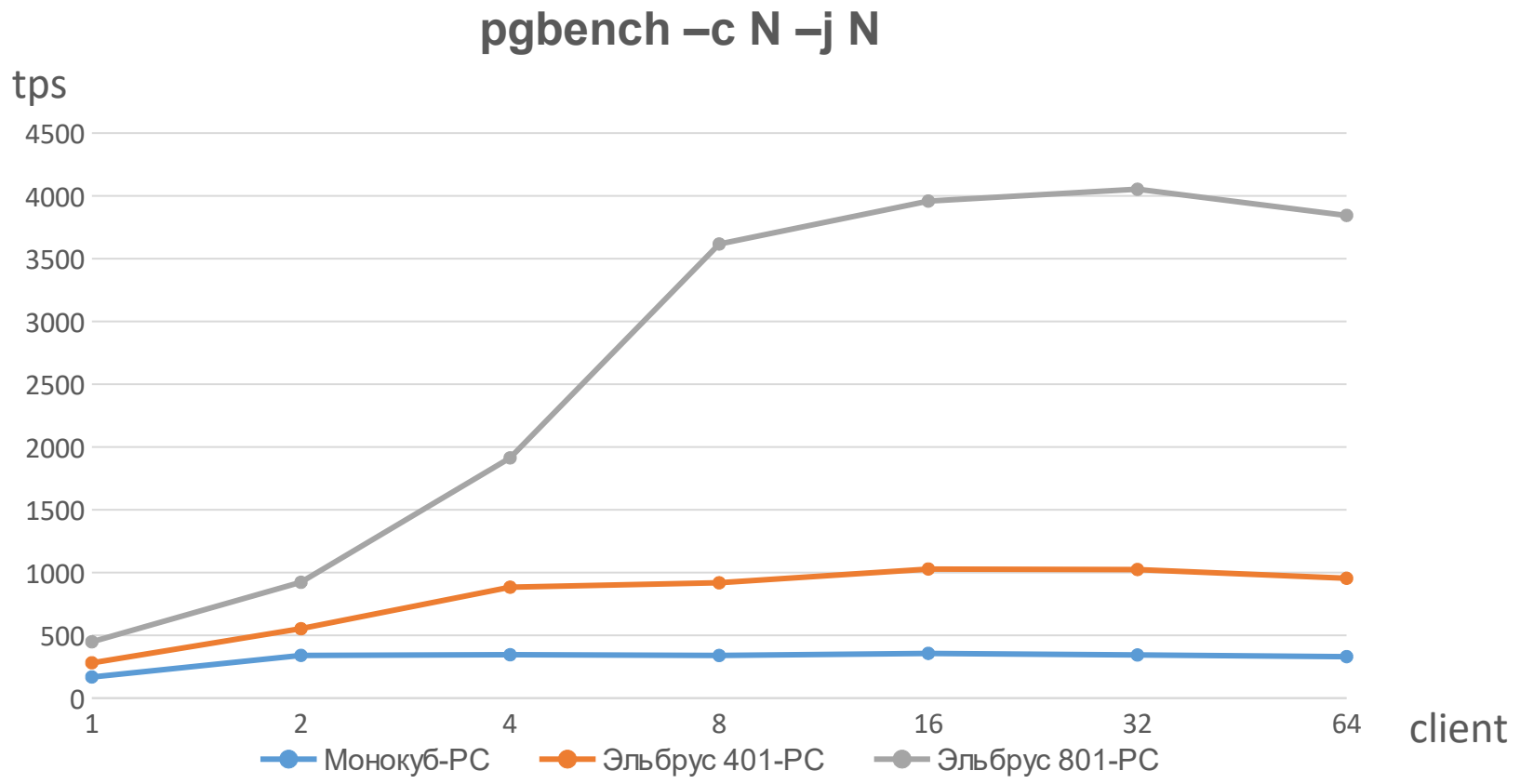
С prefetchw:

```
4,58%  postgres        [.] GetSnapshotData
3,18%  postgres        [.] _bt_compare
2,81%  libc-2.23.so    [.] _int_malloc
2,49%  postgres        [.] hash_search_with_hash_value
2,39%  libc-2.23.so    [.] __gettimeofday
2,38%  postgres        [.] AllocSetAlloc
1,99%  postgres        [.] LWLockAttemptLock
```

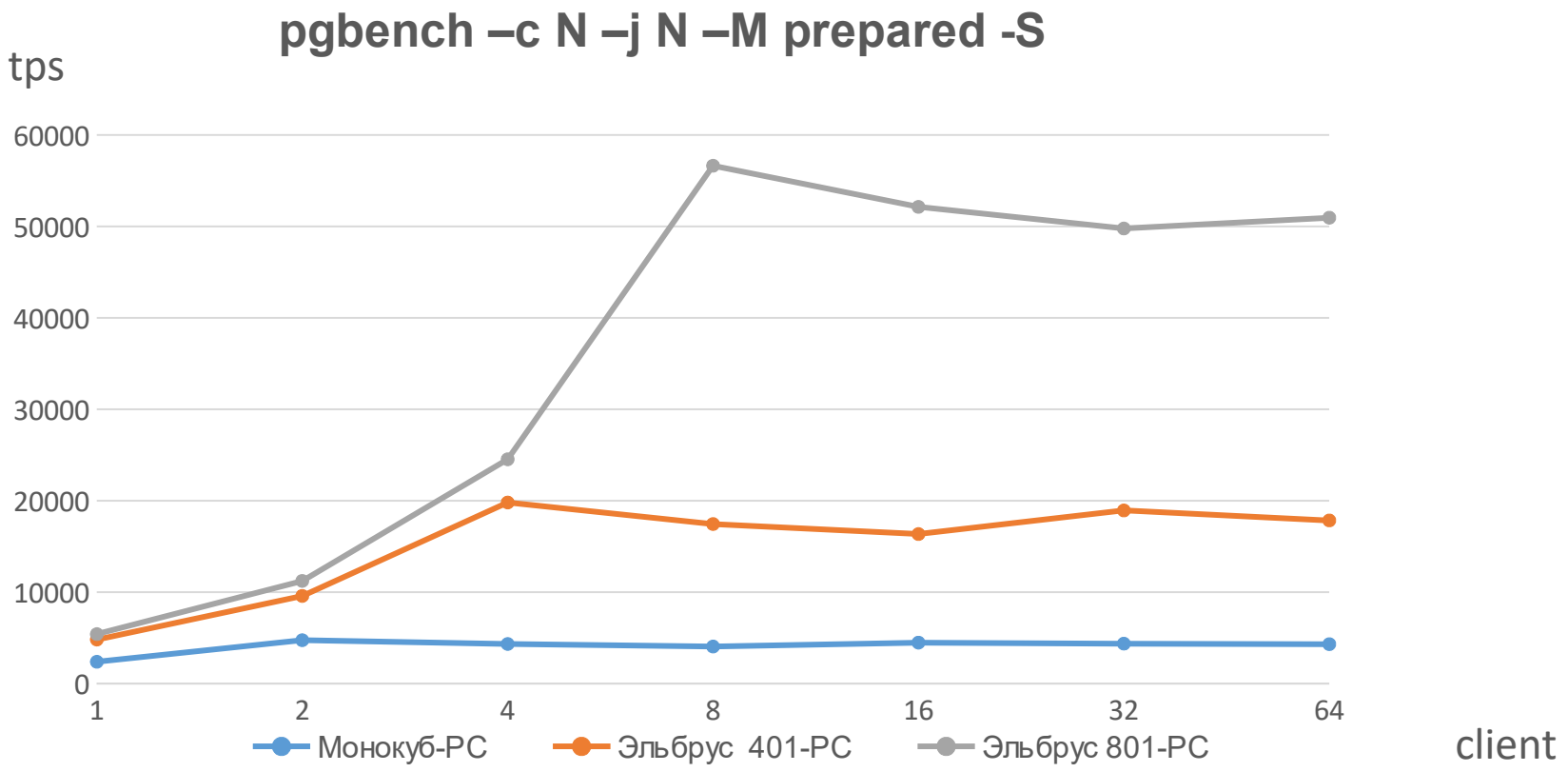
Оптимистический подход:

```
4,92%  postgres        [.] GetSnapshotData
2,96%  libc-2.23.so    [.] _int_malloc
2,94%  postgres        [.] _bt_compare
 …
1,87%  postgres        [.] SearchCatCache
1,52%  postgres        [.] LWLockAttemptLock
```

Тестирование PostgreSQL-9.5.2 на различных процессорах семейства Эльбрус

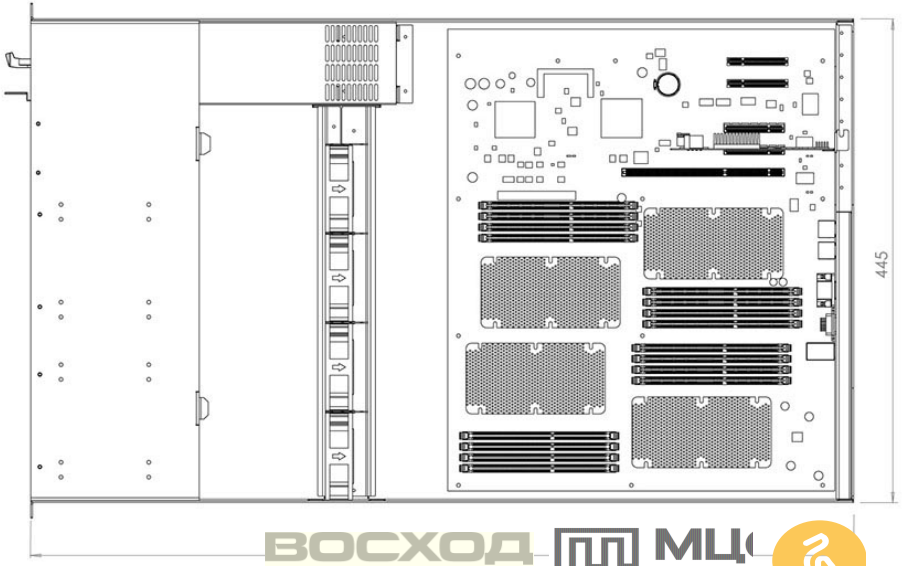


Тестирование PostgreSQL-9.5.2 на различных процессорах семейства Эльбрус

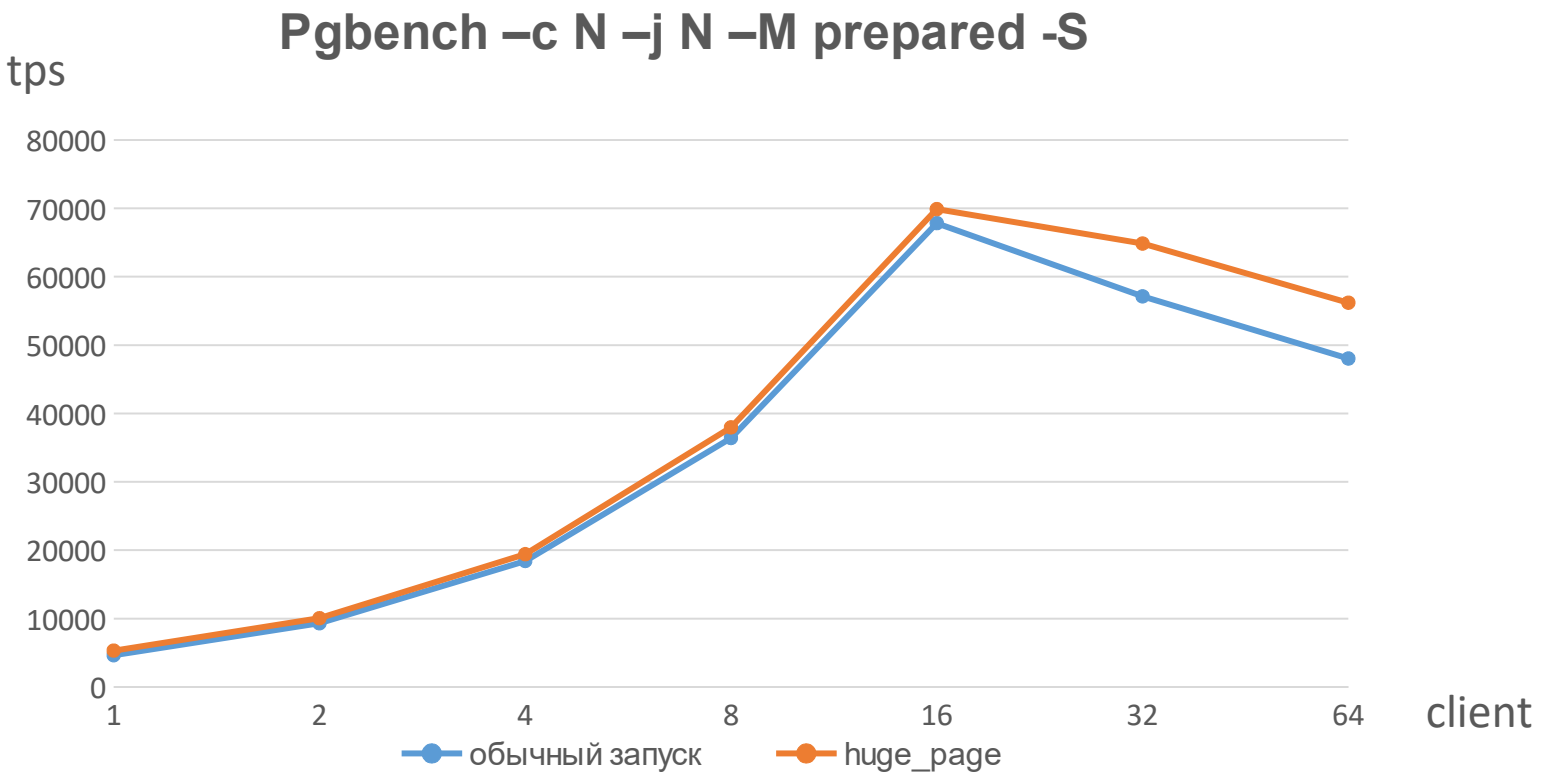


Разработка «Эльбрус-4.4 БД»

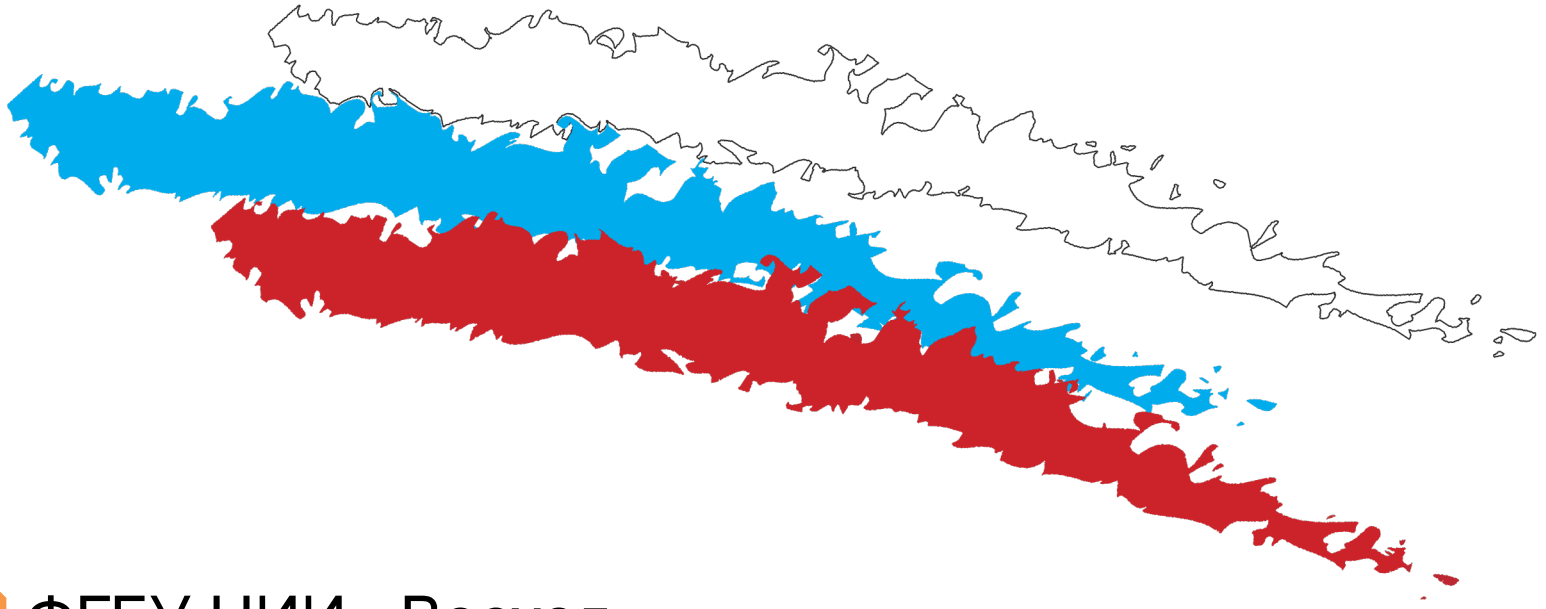
- **Кол-во дисков:** 16 дисков 2.5 с режимом горячей замены
- **Контроллер:** LSI 12 Gb/s SAS 9300
- **Чипсет:** Эльбрус КПИ
- **Процессор:** 4 процессора Эльбрус 4С, 4ядра 800MHz
- **ОП:** до 384GB DDR3 ECC
- **Питание:** 800W
- **Интерфейсы:** 2 x PCI-E 1.0
- x8, 2 x PCI
- **Сеть:** 2 x 1 Gb/s
- **Опционально:** FC, 10 Gb/s



Тестирование PostgreSQL-9.5.2 на «Эльбрус-4.4 БД»

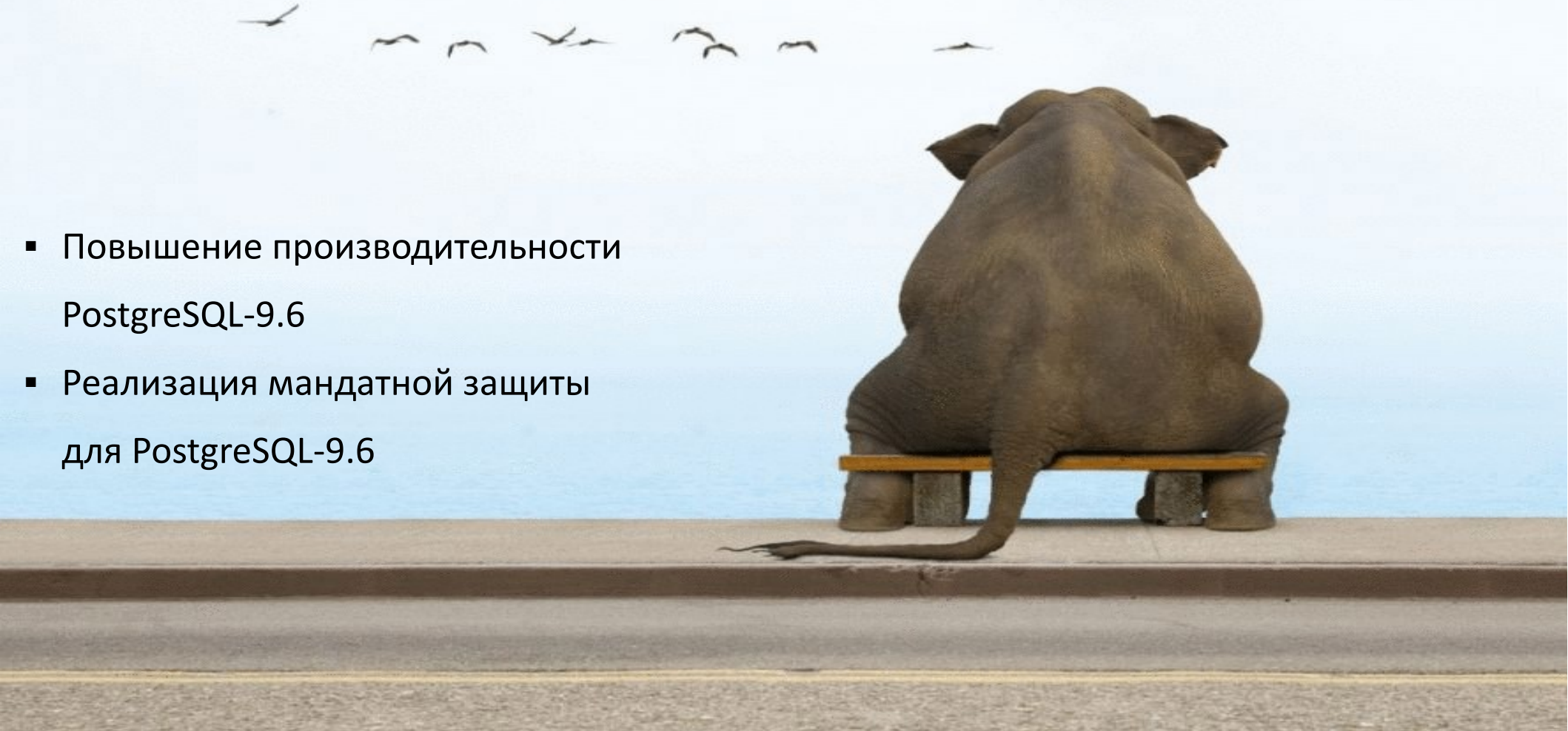


Кто использует PostgreSQL на Эльбрусе?



- ФГБУ НИИ «Восход»
- Пенсионный фонд России
- Министерство Обороны

Перспективы развития



- Повышение производительности PostgreSQL-9.6
- Реализация мандатной защиты для PostgreSQL-9.6

Спасибо за внимание!

Вопросы?

Докладчик: Королев Сергей Дмитриевич

16 марта 2017

Контакты:

Тел: +7-968-949-35-75, e-mail: Sergey.D.Korolev@mcst.ru

Богданов Иван

Главный системный администратор
ФГБУ «НИИ Восход»

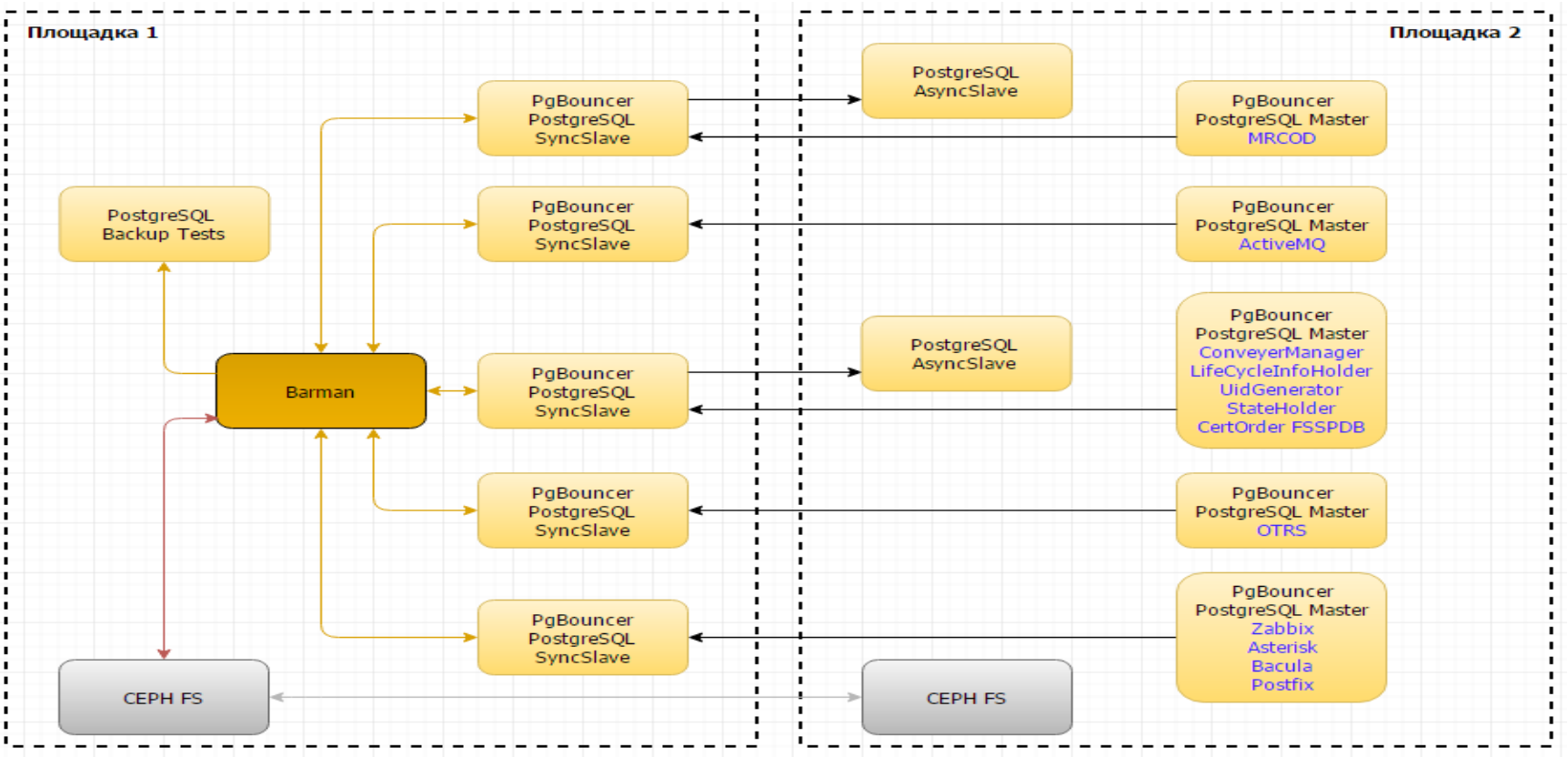
Будни администрирования PostgrSQL в ГС МИР

- PostgreSQL в ЦОД ГС МИР. Мониторинг инсталляции
- Отказоустойчивость и схемы репликации
- Особенности резервного копирования

PostgreSQL в ЦОД ГС МИР. Мониторинг инсталляции

- Мониторинг:
  - мониторинг серверов и коммутаторов
  - мониторинг PostgreSQL
  - мониторинг бизнес логики

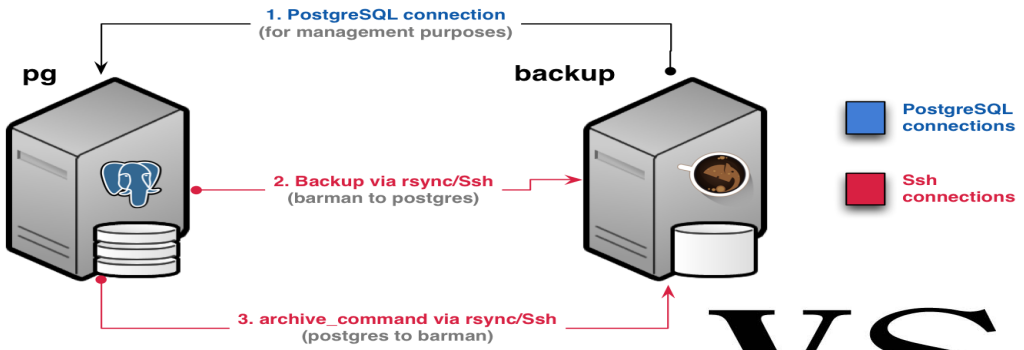
Отказоустойчивость и схемы репликации



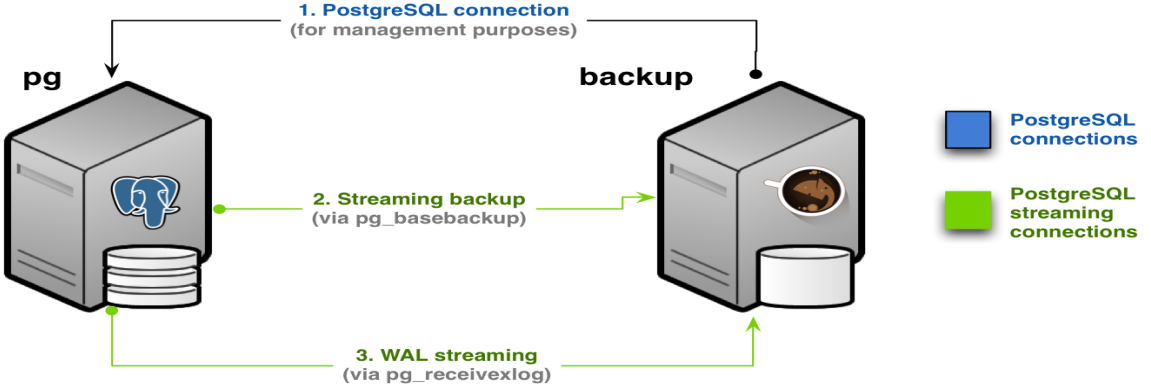
Особенности резервного копирования

- Замеры производительности (усредненные значения):
  - сеть - 195 MB/c
  - локальные дисков - 48 MB/c
  - RAID10 на SSD - 256 MB/c
  - CEPH - 129 MB/c

Особенности резервного копирования. Barman



VS



Ссылки

Шаблон Zabbix для PostgreSQL от Data Egret:

https://github.com/lesovsky/zabbix-extensions/tree/master/files/postgresql